A Network Forensics Tool for Precise Data Packet Capture and Replay in Cyber-Physical Systems

Jack Parry Daniel Hunter Kenneth Radke* Colin Fidge
School of Electrical Engineering and Computer Science
Queensland University of Technology (QUT)
Brisbane, Queensland, Australia
{ja.parry, k.radke, c.fidge}@qut.edu.au
{d4.hunter}@connect.qut.edu.au

ABSTRACT

Network data packet capture and replay capabilities are basic requirements for forensic analysis of faults and security-related anomalies, as well as for testing and development. Cyber-physical networks, in which data packets are used to monitor and control physical devices, must operate within strict timing constraints, in order to match the hardware devices' characteristics. Standard network monitoring tools are unsuitable for such systems because they cannot guarantee to capture all data packets, may introduce their own traffic into the network, and cannot reliably reproduce the original timing of data packets. Here we present a high-speed network forensics tool specifically designed for capturing and replaying data traffic in Supervisory Control and Data Acquisition systems. Unlike general-purpose "packet capture" tools it does not affect the observed network's data traffic and guarantees that the original packet ordering is preserved. Most importantly, it allows replay of network traffic precisely matching its original timing. The tool was implemented by developing novel user interface and back-end software for a special-purpose network interface card. Experimental results show a clear improvement in data capture and replay capabilities over standard network monitoring methods and general-purpose forensics solutions.

CCS Concepts

•Security and privacy → Network security; •Applied computing → Network forensics; •Networks → Cyber-physical networks;

Keywords

Network forensics; packet capture and replay; cyber-physical systems; control system security

*Corresponding author.

Thirty-Ninth Australasian Computer Science Conference (ACSC2016) February 2–5, 2016, Canberra, Australia

DOI: TODO

1. INTRODUCTION

A fundamental need for any forensics investigator or engineer trying to understand an anomaly is the ability to clearly see what happened. Having done this, and having implemented some sort of mitigation to prevent the incident recurring, the investigator's next requirement is the ability to accurately reproduce the original attack to confirm that it can no longer succeed.

These requirements are met by tools that can capture and store data traffic, and replay it later. Many such tools are available for general-purpose data communications networks. Here, however, we are concerned with the special-case of Supervisory Control and Data Acquisition (SCADA) networks as used in cyber-physical systems. Such networks offer significant challenges to the investigator because they are typically safety-critical systems which must run continually without interruption [30]. Furthermore, network traffic in such systems is subject to strict timing constraints, so any security tools used must be unintrusive.

Our own research is motivated by the need to secure the next generation of Ethernet-based electricity distribution control networks. Monitoring is required both during testing and as a continuous process in live critical infrastructure installations to allow fault diagnosis, intrusion detection, and analysis of events that resulted in serious injury or death [27]. Electricity control networks consist of numerous inter-connected, voltage-transforming substations, each one connected remotely to the company's headquarters. As part of a nation's critical infrastructure, substation design and operations are governed by strict standards, including their communications networks [14]. In particular, relevant Ethernet-based protocols, such as "GOOSE", have strict timing constraints [14].

In such an environment, a range of personnel including research and development engineers, testing and fault investigation engineers, asset management teams, and field operation engineers must be able to perform a range of network analysis tasks such as assessing packet transmission times across a network, capturing data for forensic fault analysis, and replaying data in an isolated test environment to see if anomalies are repeatable and if potential solutions are effective.

SCADA engineers have traditionally been reluctant to incorporate system monitoring and analysis tools that could interfere with safety-critical, time-sensitive data traffic. At best, a control system's Human-Machine Interaction (HMI)

software will log major system events in a 'historian' database, but engineers typically have limited ability to observe low-level network traffic. Our goal, therefore was to develop a network forensics tool which is suitable for use in safety-critical systems such as those found in electricity substations. Specifically, we needed a tool which could be used both in a "live" system to capture anomalous events and in a substation testing facility to reproduce security incidents. Therefore, our forensics tool needed to allow network traffic to be captured and replayed at speeds compatible with the "hard" real-time constraints of the cyber-physical system. In particular, all packets must be captured accurately, without affecting the original network's traffic in any way, and replayed data traffic must precisely reproduce the original timing of packets.

General-purpose Network Interface Cards (NICs) are unsuitable for this purpose because (a) they are inherently 'noisy', introducing their own data packets into the network, (b) they are controlled by standard multi-tasking operating systems and are therefore subject to arbitrary pre-emption, and (c) they have only the low-precision processor clock available for timestamping. To overcome all these problems we developed a novel solution based on a special-purpose network interface device, the Endace Data Acquisition and Generation Card [7], acting as a network tap and synchronised to GPS time. This gave us the ability to monitor network traffic silently, without introducing any extra traffic onto the network, and to both capture and insert data packets at speeds high enough to match SCADA data traffic. User interface and back-end software was developed to allow straightforward capture and replay of data traffic, inline Berkeley Packet Filtering (BPF), as well as simple packet modification and filtering operations during replay. Experimental results show that the resulting tool compares very favourably with attempts to use standard networking tools for SCADA system forensics.

2. PREVIOUS AND RELATED WORK

Our concern in this research was supporting security-critical SCADA network analysis by capturing and replaying network traffic with high timing accuracy. There are, of course, many existing tools for performing packet capture, or "pcap", operations in communication networks.

Primitive libraries of functions such as *libpcap* provide basic network interfacing, packet capture and filtering capabilities [18], and are the basis of many software utilities. Building on this, command-line packet analysers such as *tcpdump* support capture and analysis of network traffic [19]. Such tools can capture packets from a Network Interface Card or analyse previously-captured "pcap" files. As command-line tools, they can be incorporated into other tools and scripts. For instance, for users wanting a Graphical User Interface, popular tools such as *Wireshark* add the ability to visually examine captured packets [24].

Apart from observing traffic, there are also general networking tools for generating or "injecting" network traffic onto a communications link. For instance, *Tcpreplay* is "a suite of free, open-source, UNIX utilities for editing and replaying previously captured network traffic, originally designed to replay malicious traffic patterns to intrusion detection systems" [3]. It allows network traffic captured and stored in local files by tools such as tcpdump to be reinserted into the network, thereby allowing previously-seen

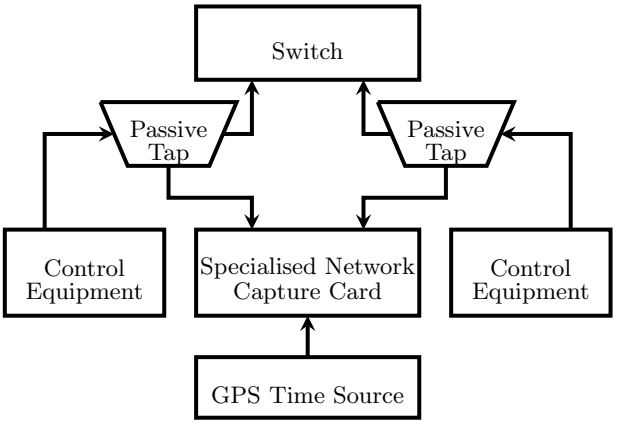

Figure 1: Physical architecture of our SCADA network forensics system.

security attacks to be reproduced at will. Like tcpdump, it is a command-line tool and is designed to work with standard network cards [3]. Another such tool is *Nemsis*, "a command-line network packet crafting and injection utility well-suited for testing Network Intrusion Detection Systems, firewalls, IP stacks, etc" [22].

A central concern of our research is to be unintrusive, in order not to affect the required timing of packets in the SCADA system. For instance, the GOOSE protocol used in electricity substations requires a packet latency no greater than 3 milliseconds [14]. Some existing packet capture and injection tools such as *pktd* are similarly designed to meet strict performance goals, but in this case the issue is to minimise the tool's footprint on its host computer [9], rather than a concern with affecting the timing of data packets on the network. Another early tool with goals similar to ours was *nCap* which aimed to overcome the limitations of general-purpose operating systems for capturing high-speed traffic by using proprietary network adapters [6]. More recently, commercial tools such as *Sniffer10G* have provided support for high-speed capture and replay of Ethernet traffic [21].

None of these tools, however, were specifically designed for SCADA network traffic, as needed for diagnosing anomalies in a cyber-physical control system. In particular, tools relying on standard Network Interface Cards are potentially noisy, do not guarantee to preserve the order of packets, nor do they have precise timing capabilities. While none of these properties may be crucial in a general-purpose network, where there are no "hard" real-time constraints and packets routinely arrive at their destination out of order, in critical infrastructure obtaining and timing packets accurately and in their original order is essential [10]. Our approach solves all of these problems.

3. HARDWARE DESIGN

In this section we present the options and decisions that led to the physical architecture of our solution (Figure 1). Section 4 then describes its software implementation and Section 5 analyses the overall system's performance.

3.1 Access points for network captures: Inline taps versus mirror ports

Our first concern was the need to capture all network traffic, without losing any data packets. There are two main options for network traffic capture, inline taps and mirror (SPAN) ports [23]. Inline taps offer greater certainty that all network traffic is acquired, because mirror ports may drop malformed packets, or may stop mirroring data altogether if switch computation buffers fill [17]. There are also problems with mirror porting if Virtual Local Area Networks (VLANs) are used, because the mirror ports need to be configured to recognise VLAN traffic. Packets with VLAN tags that do not match the port settings are dropped. VLANs are a recommended method of segmenting networks in critical infrastructure [16] and are common in critical infrastructure, so this issue is significant. Further, switches will inject their own packets and may modify existing packets, especially to change their priority [5]. For all of these reasons, a mirror port may not be passive [11], so for critical infrastructure purposes inline taps were our preference.

(On the other hand, an advantage of mirror ports is that they can be activated as needed, whereas installing an inline tap requires breaking the communications link. We have assumed, however, that the necessary taps will be installed permanently as part of the critical infrastructure system.)

3.2 Network capture method: Network interface controllers versus data capture cards

Regardless of how raw data is accessed, via inline tap or mirror port, the network packets need to be captured somehow. If a standard Network Interface Controller (NIC) card is used packets may be dropped due to being oversized, malformed, too plentiful, etc. Also, packets may be reordered from the sequence they appeared on the network due to the multi-tasking nature of the host operating system [29]. Furthermore, packets captured by a NIC are timestamped by the host processor's low-precision clock [1].

To overcome these inherent limitations with NICs, our solution is to instead use a specialised data capture card, with its own onboard packet timestamping capability, and facilities for synchronising to a high-precision timing reference such as a GPS clock.

3.3 Physical architecture

Taking both points above into account, the overall physical architecture of our data capture and replay solution is shown in Figure 1. Given two pieces of control equipment communicating via a switch, we place passive inline taps (e.g., optical splitters) into the network and these are connected to a specialised network capture card with onboard timestamping and high-precision time synchronisation capabilities.

This setup has a number of interesting secondary benefits beyond capturing traffic accurately, such as the ability to measure the time a packet takes to traverse a network. Measuring packet transmission times to a high accuracy means that critical infrastructure research and development engineers can ensure that constraints on packet transmission times specified in relevant international standards for critical infrastructure [14] are met.

To instantiate the architecture in Figure 1 we used the following equipment.

Control equipment As examples of typical Intelligent Electronic Devices (IEDs) used in critical infrastructure control systems we used a General Electric C60 Breaker Management Relay [8] and a Schweitzer Engineering Laboratories SEL-421 Protection Automation Control [25].

GPS time source As the high-precision time source we used a Tekron TCG 01-G GNSS Timing Generator [26].

Specialised network capture card To create the specialised data capture card with onboard processing, timestamping, and time synchronisation capabilities, we used an Endace Data Acquisition and Generation Card 7.5g4 [7].

3.4 Validation

To arrive at the architecture shown in Figure 1 we performed a series of tests, described in Section 5. These tests highlighted the quality of network capture timing in different configurations. We began by using two computers with standard NICs synchronised using the Network Timing Protocol (NTP). We then progressed through NICs synchronised using the newer Precision Timing Protocol (PTP), and finally specialised network capture cards synchronised with one pulse per second (1pps) and PTP.

In our final design, the Endace Data Acquisition and Generation Card [7] we used has an onboard clock, which we synchronised to a GPS time source, so that packets could be timestamped at a very high fidelity. The particular card used has a claimed timing accuracy of 7.5ns. Further, the card has onboard storage such that no packets are lost on any of its four gigabit interfaces. It also has a Field Programmable Gate Array (FPGA) onboard, which could be programmed to do small computational tasks, such as Berkeley Packet Filtering (BPF).

3.5 Replaying data

After anomalous network traffic has been identified and recorded, a forensics investigator may want to replay the event. This allows potential mitigations to be tested. To this end we investigated standard packet replaying tools. We assessed the most commonly-used such tool, *tcpreplay* [3], and showed it to suffer from cumulative timing drift, as well as timing inaccuracy. By contrast we also tested our own implementation using the Endace DAG card and showed that the cumulative drift is eliminated, and the timing of replayed packets is in the order of 1000 times more accurate.

Furthermore, it is sometimes necessary when replaying packets from a stored packet capture file in an isolated testing environment to change the source and destination addresses in the packet headers to allow them to be transmitted in the different physical setup. To support this we developed a graphical interface for critical parts of *tcprewrite* and *tcpprep*, which are Linux-based pcap manipulation tools. Using our user interface, forensics investigators can update MAC and IP addresses for some or all of the packets in a captured network packet file.

3.6 Active tap capability

As noted above, in a critical infrastructure environment it is essential that *passive* inline taps are used. One reason for this is that active taps require the presence of power, so failure of an active tap will also compromise packet transmission in the network being monitored. Nonetheless, in an isolated testing environment *active* inline taps may be acceptable, so

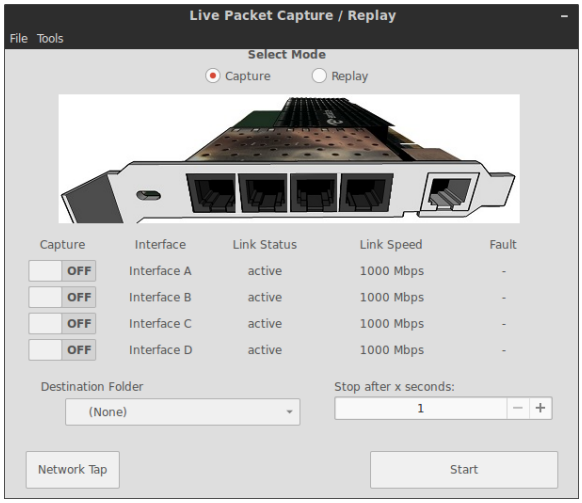

Figure 2: Initial user interface in 'capture' mode.

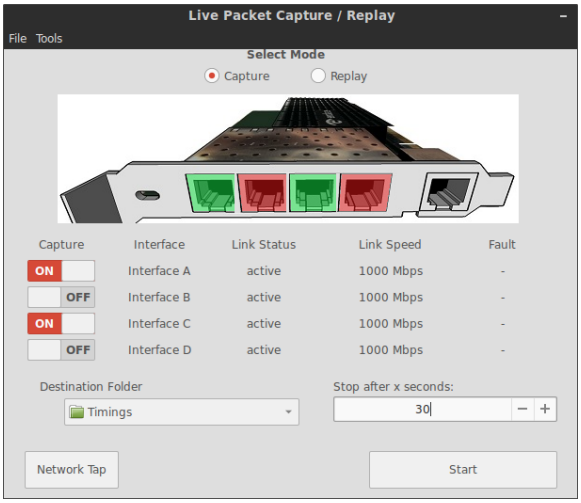

Figure 3: User interface in 'capture' mode with selections made.

we also measured how much delay such an active inline tap adds to the network, using our Endace implementation.

This was possible by using a combination of our capture and replay capabilities, with the user interface providing the user with the ability to use the four-port Endace card as two inline taps. Each pair of two ports can be used as an input and an output, with the packets being captured and timestamped as they enter the card, before being retransmitted.

4. SOFTWARE IMPLEMENTATION

In this section we describe the capabilities of the user interface and back-end software we developed to support the hardware architecture presented in Section 3. The main features required were capturing traffic, replaying traffic, modifying captured packets, and providing an inline tap mode.

4.1 Approach

Software development was done in C using low-level API calls for the specialised network capture card [7]. Initially this resulted in a prototype program involving thousands of lines of code. It provided low-level control over network traffic, but had persistent problems such as API calls failing. It was found that the card's manufacturers recommend using a serialisation wrapper for calls to the API, and the prototype code thus became quite complex, even without error checking. Instead, therefore, we reverted to using the manufacturer's high-level API functions for capturing data packets, which resulted in the core non-user-interface code for capturing packets being only a few hundred lines long. The user interface for capturing packets was approximately 400 lines of code. Capturing packets was one of four main areas of the developed tool, which also included an inline tap capability and user interfaces for tcpprep and tcprewrite functionality.

An error in the provided card interface was encountered, involving termination of the inline tap once a buffer had successfully been written to disk. After examining the manufacturer's source code, a logical error was identified such that the thread responsible for writing to disk did not have a valid way of terminating after successfully clearing the buffer. This error caused the thread to loop every millisecond indefinitely. To solve this problem, we ran the inline tap capability in a separate thread terminated by the main application upon completion.

4.2 Initialisation

Figure 2 shows the network capture user interface when our tool is run. Prior to the graphical interface being displayed to the user, the following series of steps are executed.

1. Drivers and firmware for capture and replay are loaded.
2. The card is reset to its defaults, with port auto negotiation turned on.
3. 128MB of system memory is allocated to the card, and the tool is run in a mode which uses all of the available CPU resources of a single thread (to aid in guaranteeing no packet loss).
4. The number of ports on the card are counted and the port status checked for each.
5. The number of supported transmit streams are detected.

After the above sequence, the graphical interface is updated to reflect the capabilities of the installed card and user interaction begins.

4.3 Capturing traffic

Figure 3 shows the network capture user interface. At the top of the window an image of the packet capture card is displayed. Which ports to capture from can be turned on or off via the buttons on the left hand side and these selections are displayed visually. At the bottom on the left is the folder in which to save the captured traffic, and on the right is the duration in seconds of the packet capture.

Once the user selects "Start" in the bottom right, the following sequence of steps are performed.

1. The DAG card is reset to its defaults and the capture buffer is cleared.

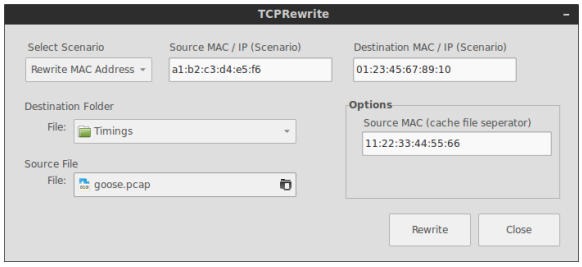

Figure 4: tcprewrite graphical user interface, showing modification of all source and destination MAC addresses, for packets from a specific MAC address.

2. The maximum length of each packet is set to 9600 (the maximum supported by the card), so that oversized packets will be captured.
3. Variable length packets are enabled and packet padding is disabled.
4. As per the ERF file format used by the card, alignment parameters are set to allow future replay of captured packets.
5. Network interface ports are enabled/disabled based on the selections made by the user.
6. The capture time and destination folder are set based on the selections made by the user.
7. A unique output filename is created using the current date and time.

At the completion of the packet capture, a "Capture Complete" popup is presented to the user.

4.4 Replaying traffic

The interface for replaying traffic is very similar to the interface for capturing traffic, shown in Figure 3. The difference is the selection of "Replay", instead of "Capture", at the top of the window. The packets are replayed in the same order, and with the same inter-packet delays, as recorded in the file to be replayed.

Packet capture and replay were validated. Firstly, packet capture was validated using *NetworkMiner*, a forensic analysis tool [12]. However, since the output of our tool is in the ERF format used by the network card, which is not recognised by NetworkMiner, *editcap* [28] with encapsulation type set to Ethernet frames was used to convert '`.erf`' files to '`.pcap`' format without ERF headers. The contents of the captured file were found to be an accurate capture of the data that was sent. For replay, NetworkMiner was not suitable for comparisons based on timing. Initial comparisons were made using Wireshark, and detailed analysis was completed using Python scripts and C code. For detailed experimental results on packet generation and capture, see Section 5.4.

4.5 Modifying captured traffic

To replay traffic captured in a live system in a separate test environment requires updating hardware Media Access Control (MAC) or Internet Protocol (IP) addresses for the new environment. To support this capability for system engineers users who are not familiar with Linux command-line tools, we implemented a user interface for critical parts of *tcprewrite* and *tcpprep* [4, 2]. An example of one of the interfaces provided is shown in Figure 4. In this example, the user is searching for all packets with a source MAC address of 11:22:33:44:55:66 and is setting all source MAC addresses for those packets to a1:b2:c3:d4:e5:f6 and all destination MAC addresses for those packets to 01:23:45:67:89:10.

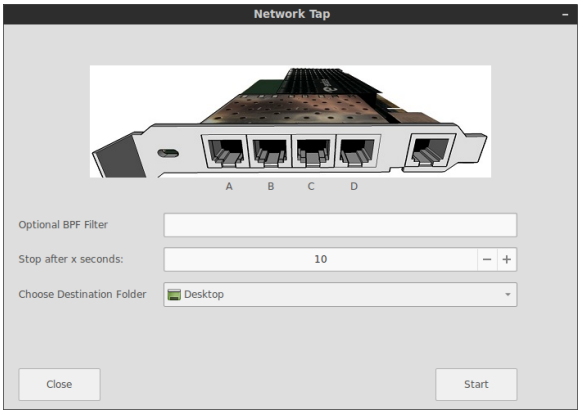

Figure 5: Inline tap user interface.

4.6 Inline network tap mode

For use in an isolated testing environment, where interfering with network traffic is not an issue, we allowed the specialised network capture card to be configured as an active inline tap, thereby allowing packets to be manipulated "on the fly". To do this, bidirectional forwarding between two ports, while capturing the traffic, was required. Although the simplest way to achieve this may appear to be copying the input buffer from one port directly to the output buffer of another port, this only provides the forwarding capability and not the capture capability. To capture traffic at the same time, the stream from buffer to buffer needs to be duplicated. The user interface for the inline tap is shown in Figure 5.

Once inline tap mode was working for a pair of ports, the functionality was expanded to create a dual inline tap on the four port card. The two taps are marked in the generated ERF file. The Endace card is manufactured in two-port and four-port versions, so we allowed for the possibility of using the network card to act as either one or two inline taps.

An optional Berkeley Packet Filter (BPF) can be applied when using this inline tap mode. For example, "`icmp`" may be entered on the user interface to only forward packets past the inline tap that are of the ICMP protocol. However, this matching is performed on the host computer's CPU, rather than on the network capture card, and as such a latency measured in milliseconds is added if the BPF functionality is used.

5. EXPERIMENTAL RESULTS

In this section we analyse the performance of our combined hardware (Section 3) and software (Section 4) solution to the problem of cyber-physical network capture and replay.

Our experiments firstly identified the timing accuracy of sending and receiving under different configurations.

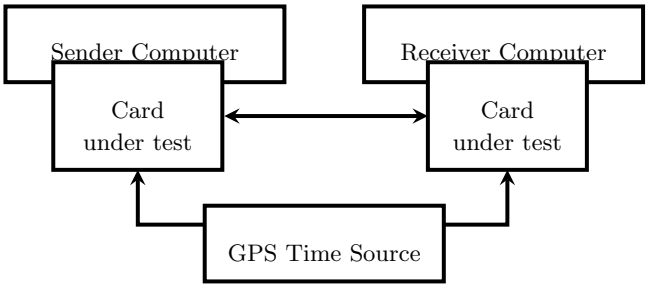

Figure 6: Generic physical experimental setup.

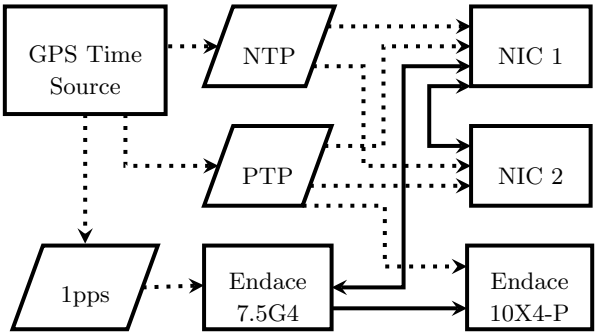

Figure 7: Protocol communication flows in the experimental setup.

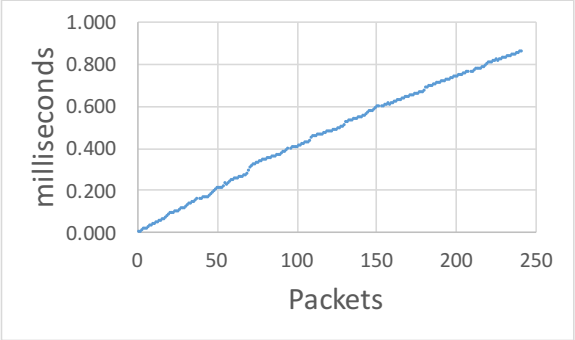

Figure 8: Average cumulative error of the duration between when the first packet is sent and when the nth packet is sent compared with when the first packet is received and the nth packet is received, with packets sent using tcpreplay for 100 NIC-to-NIC tests synchronised using NTP.

These included typical state-of-the-art configurations currently used in critical infrastructure development and fault finding, including using standard NIC interfaces to capture data and to generate or replay data for testing. At best, these NICs will be synchronised using either the standard Network Timing Protocol or the newer and more accurate Precision Timing Protocol. The standard tool, *tcpreplay*, was used to replay the test data for these typical setups. These experiments provided a baseline, and also provided evidence of a flaw in using tcpreplay. Using these baselines, we then showed that our solution is in the order of 1000 times more accurate, and also overcomes the identified flaw in tcpreplay.

Having demonstrated that we can send and receive at a very high accuracy, measured in nanoseconds, we performed two further sets of experiments. Firstly, we used our developed tool in its *active* inline tap mode. The process of acquiring data from a network on one port, and putting the data back onto the network on another port, inevitably takes time. Knowledge of just how much time this takes is essential for anyone using such an inline tap. Finally, with the knowledge of how long our active inline tap solution takes, we then used two active inline taps to timestamp the same packets at different points in a network, thereby demonstrating the ability to use our tool to record how long packets take to traverse a network, to a known accuracy.

5.1 Experimental setup

Our generic experimental setup is shown in Figure 6. This configuration sometimes changed slightly depending on the test. One computer was running a Debian 7-based distribution of Linux and the other was running running CentOS v7.0. CentOS is a free enterprise operating system based on the Red Hat version of Linux, with Red Hat being the version of Linux commonly used when Linux is required in critical infrastructure, due to its support. A protocol-oriented depiction of our experimental setup is shown in Figure 7.

In each experiment, a packet capture file, with known timings, was replayed through one network card and captured on another network card. The time when a packet was captured was compared with when it was sent, both as an absolute value from the test's beginning (e.g., packet 2000 should have arrived 8 seconds after the first packet was sent) and as a relative value between consecutive packets (e.g., packet 2000 should have arrived 0.001s after the 1999th packet was sent). All experiments were run 100 times and their results averaged, such that the data presented in the graphs, starting with Figure 8, has each point as the average of 100 tests. Non-averaged results are discussed in each section. Where tcpreplay was used, the version used was 4.1.

5.2 NIC to NIC, synchronised using NTP, sending using tcpreplay

In this experiment, we used *tcpreplay* to send packets from a standard network interface controller (NIC 1 in Figure 7) to another network interface controller (NIC 2) which we observed using *Wireshark*. We synchronised the sender and the receiver using NTP [20] generated from our GPS clock, using secondary NICs in each machine. This is the standard setup most test environments would use to replay data. We therefore used this test as a baseline to identify issues with sending and receiving. The error between when a packet is sent, and when it is received, is shown in Figure 8.

Note the obvious cumulative error shown in Figure 8, as well as the individual packet errors, when tcpreplay is used. For further discussion and more results from the NTP test, see Section 5.3.

5.3 NIC to NIC, synchronised using PTP, sending using tcpreplay

This experiment was the same as that in Section 5.2 except the two NICs were synchronised using PTP [13]. The Precision Timing Protocol is an enhancement on NTP, pro-

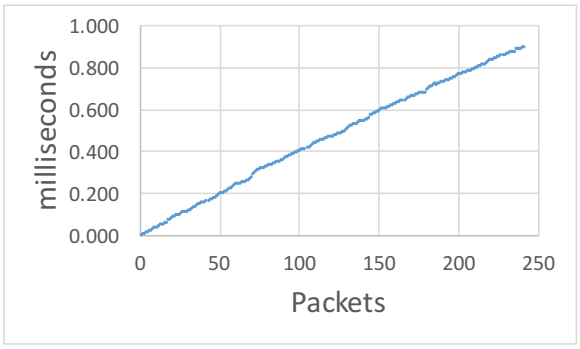

Figure 9: Average cumulative error of the duration between when the first packet is sent and when the nth packet is sent compared with when the first packet is received and the nth packet is received, with packets sent using tcpreplay for 100 NIC-to-NIC tests synchronised using PTP.

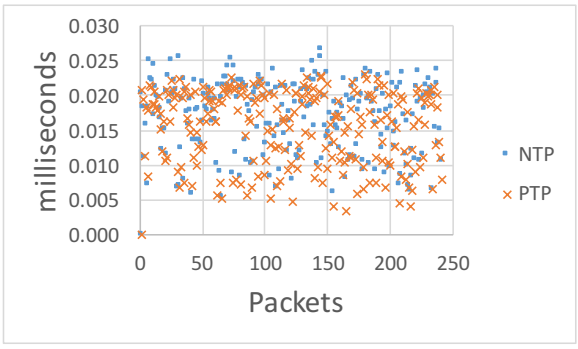

Figure 10: Comparison of inter-packet errors in the NTP and PTP NIC-to-NIC experiments of Figure 8 (NTP) and Figure 9 (PTP).

viding sub-microsecond accuracy [13]. The error between when a packet is sent, and when it is received, is shown in Figure 9.

Figures 8 and 9 clearly demonstrate the cumulative nature of the error between when packets are received compared with when they should have been. This error became quite significant as our tests involved sending 241 packets over 9.22 seconds; for the last packet the error was almost a millisecond on average.

It seems that tcpreplay measures the delay to add between each packet, and adds that delay to the time after the last packet was sent, thus aggregating (correctly) all the delays and adding (incorrectly) all of the time taken to send each of the packets. To further test this issue with tcpreplay, we replayed data out of one NIC and back into a secondary NIC on the same computing device (thus omitting issues with time synchronisation), whilst still synchronising the device to the GPS signal using PTP through a tertiary NIC. In the case the aggregating nature of the absolute error when using tcpreplay to replay packets was still clearly evident. Further, we replayed using tcpreplay and captured using our Endace solution to rule out the possibility that the source of the error was data being captured via Wireshark or tcpdump. In this case the same error was still present. The cumulative error was only removed once tcpreplay was replaced with our software.

Having identified the cumulative error when using tcpreplay, we then analysed the data again to identify any error in the durations between consecutive packets. The results of this analysis, for both NTP and PTP, are shown in Figure 10. Each point in Figure 10 is the average for 100 tests. The greatest error margin for a single duration between two consecutive packets when sending tcpreplay and synchronised using NTP was 206μs and for PTP was 369μs. However, on average PTP had a better inter-packet error at 15.5μs compared with NTP's average error of 17.1μs. Nevertheless, as shown in Figure 10, since both NTP and PTP are software timestamped, and CPUs are multitasking, regardless of which is used to synchronise the sender and the receiver there are significant timing variations with both PTP and NTP.

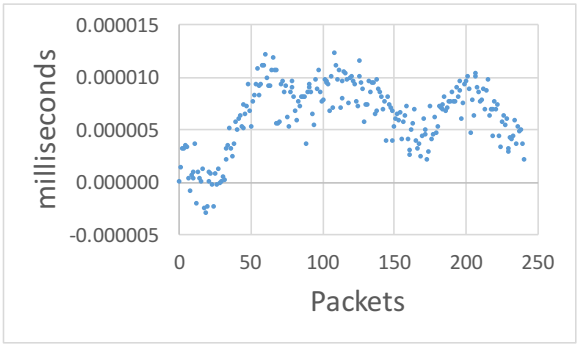

Figure 11: Average error in the interval between packets at the sender versus the interval between packets at the receiver, with packets sent using our forensics tool for 100 Endace-to-Endace tests synchronised using 1pps and PTP.

5.4 Endace to Endace, synchronised using 1pps and PTP, sending using our solution

Having measured the performance of standard solutions (Sections 5.2 and 5.3), we then started testing our own solution (Sections 3 and 4). In this test we replayed using calls to the Endace drivers rather than tcpreplay, and captured using the specialised network capture card and our own software rather than a NIC. One of our two Endace cards (an older model) used a 1 pulse per second (1pps) signal from our GPS clock, while the other Endace card used a PTP signal from the GPS clock. Using this testing method, there was zero cumulative error, unlike the tcpreplay tests. The results of this experiment are shown in Figure 11.

Since there was no error accumulation in the results of this test, unlike those shown in Figure 8 (NTP) and Figure 9 (PTP), we can immediately compare the results in Figure 11 with those in Figure 10. It should be noted that the results of this test were in the order of 1000 times better, with most errors being less than 10ns on average in Figure 11, compared with 10μs in Figure 10.

Further, analysing the data individually revealed that the worst error for the delay between two packets experienced by our solution was 46ns, compared with 206μs and 369μs for NTP and PTP, respectively, over NICs. The average value for the error in the delay between two packets using

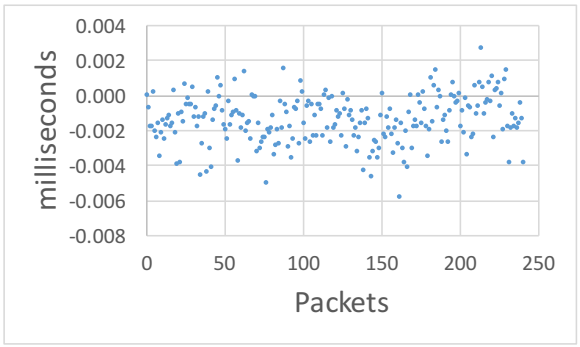

Figure 12: Error introduced due to recording from a NIC.

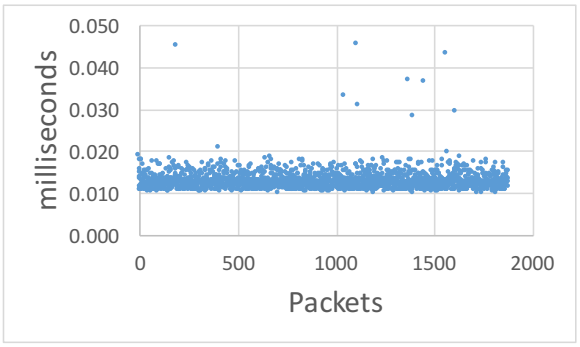

Figure 13: Additional time added in inline tap mode using our solution.

our software and the Endace cards was 8.9ns.

We used standard tolerance interval statistical calculations for a mean of 8.9ns and a standard deviation of 7ns, across our 240 inter-packet intervals and 100 tests to show that with 99% confidence, 99% of the inter-packet interval errors will be within −10ns to +27ns.

5.5 Endace to NIC, synchronised using 1pps and PTP, sending using our solution

Having demonstrated the accuracy of the Endace to Endace solution in the test described in Section 5.4, we replaced the receiver with a standard NIC, to demonstrate the issue of recording either from a mirror port or from an inline tap, using a standard computer-hosted NIC. The results of this experiment are shown in Figure 12.

Having seen that the maximum error from the experiment in Section 5.4 was $46ns$, and the tolerance interval was −10ns to +27ns, we see in Figure 12 that by replacing the receiver with a standard NIC the average error increases to 17μs and the maximum individual inter-packet interval error became 234μs. Hence, almost all of the error is due to the NIC itself, further demonstrating the weakness of using a standard NIC to record data in time-critical network.

5.6 Endace to Endace, synchronised using 1pps, active inline tap duration

In the experiment described in Section 5.4 we showed that sending and receiving between the specialised network cards can be done with nanosecond accuracy. In a test environment, our solution can be used as an active inline tap, using our software and by physically connecting the network cable into one port of the network card, and out of the second port on the network card. The active inline tap means that packets can be completely captured (no packet loss) with timestamping more accurate than using a NIC on a computer connected to either an inline tap or mirror port (as demonstrated by the results described in Section 5.5).

In this experiment we ran 100 tests with ports 1 and 2 on the specialised capture card as an inline tap, and then used the output from port 2 as the input to ports 3 and 4 which were also used as an inline tap. The difference between the timestamps on packets passing through ports 1 and 2, compared with when they passed through ports 3 and 4, was the additional time that using the specialised hardware as an active inline tap adds to the transmission time of packets. The results of these experiments are shown in Figure 13.

The experiments showed that the addtional time added to the end-to-end transmission time of packets, by having the active inline tap in the network path, was in the order of 12μs. While this compares favourably with the 17μs added by recording using a standard NIC, it should be noted that using the specialised network capture card in this configuration is only meant as a convenient testing tool, in a laboratory setting. As shown in our intended architecture in Figure 1, a passive inline tap (e.g., an optical splitter), which adds negligible additional time, should be used in a "live" system, and then the specialised network capture card can record from the passive inline tap with low nanosecond accuracy.

5.7 Endace to Endace, synchronised using 1pps, transmission duration

Having shown in the experiment described in Section 5.6 how much overhead inserting our tool adds to the end-to-end transmission time, we then used two instances of our tool, synchronised using a GPS clock, to timestamp the same packet at two different points in a network. This provides a measurement, to a known accuracy, of the time taken for a packet to traverse a network. End-to-end packet transmission duration is essential knowledge in a time-sensitive environment such as a critical cyber-physical system. In particular, some protocols in critical infrastructure have transmission time limits, such as the GOOSE protocol's 3ms limit as specified in IEC Standard 61850 [15].

In this experiment the passive taps shown in Figure 1 were replaced with the Endace card acting as an active tap. The control equipment IEDs were as specified in Section 3.3, and the GE-C60 and SEL-421 devices were configured as a GOOSE publisher and subscriber pair. Four switches were placed between the two IEDs. The inline taps were placed between each IED and its corresponding edge switch. The results of this test, capturing traffic for 30 minutes, resulted in 1797 packet durations, as shown in Figure 14.

Figure 14 immediately provides two insights useful in a testing environment. Firstly, the mean (average) transmission time was 1.168ms but this figure is severely skewed by the three outlying values. It should be noted that Figure 14, has a logarithmic y-axis for the time value to accommodate the outlying values which included a maximum value of 177.024ms. Since these three values, out of the 1797 packet durations, were so different from the rest of the values, median and mode values were used to provide clarity

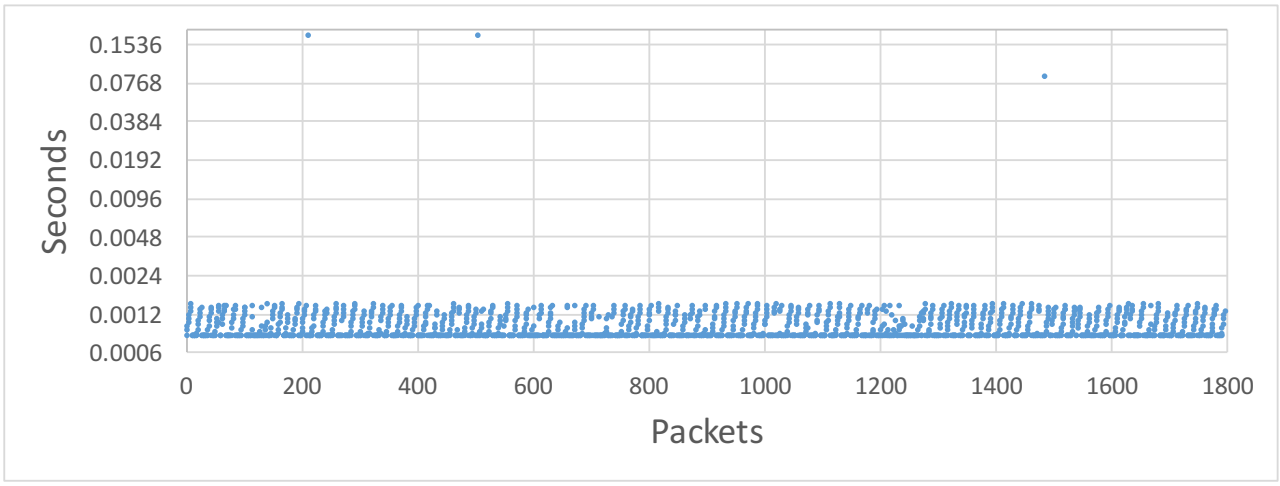

Figure 14: GOOSE packet transmission time across a four switch network using an active inline tap.

regarding "typical" values. The median (middle) transmission time was 0.784ms and the mode (the value that occurs the most) transmission time was 0.784ms. Therefore, ignoring the small number of outliers, Figure 14 tells us that the typical transmission time across the network is approximately 0.784ms.

The second observation of significance, which demonstrates the usefulness of this tool in a test environment, is the bursty nature of the traffic as indicated by the saw-tooth pattern of the graph. Such a graph would be useful to a network designer as it suggests that one or more of the switches creates a buffer before releasing all of the accumulated data at once. The active inline tap could then be easily moved throughout the network to identify which devices are causing the bursty traffic. Similarly, the inline tap could also be moved throughout the network to identify the sources of delays, if they are repeatable, of the significantly outlying values for transmission time duration. Such analyses would facilitate potential remediation of issues, with the effectiveness of potential solutions also being testable using our tool.

6. CONCLUSIONS

The ability to capture all network traffic and replay it accurately is essential for network forensics investigations. Our research was motivated by the need for such a tool to use in power distribution substations (and their testing facilities). Due to their hard real-time constraints, such cyber-physical systems introduce additional challenges beyond those found in general-purpose networks, making traditional solutions based on Network Interface Cards inadequate.

Here we have described a hardware-software solution that builds on a special-purpose network interface card to produce a tool which (a) guarantees to capture all data packets, and preserve their original sequencing, (b) during data capture does not interfere with the network traffic in any way, and (c) allows the captured traffic to be replayed precisely, including its original timing. Installed in an appropriate configuration, this tool can support unintrusive monitoring of "live" control system data traffic and reproducible experimentation with security incidents in an isolated testing facility.

A possible extension of this system would be to allow data capture at the bit level (1s and 0s), rather than whole packets only, which would be useful for forensics investigators exploring malformed network traffic. However, our experiments show that the firmware in the current Endace card used only allows capture of "packets", as opposed to "bits", so new firmware needs to be developed to support this extension.

Acknowledgments

This research was funded in part by ARC Linkage Project LP120200246, *Practical Cyber Security for Next Generation Power Transmission Networks.*

7. REFERENCES

[1] L. Abeni, A. Goel, C. Krasic, J. Snow, and J. Walpole. A measurement-based analysis of the real-time performance of Linux. In *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, 2002.*, pages 133–142. IEEE, 2002.

[2] AppNeta. Tcpprep: Pcap pre-processor, 2015. http://tcpreplay.synfin.net/wiki/tcpprep, accessed 11 August 2015.

[3] AppNeta. Tcpreplay: Pcap editing and replaying utilities, 2015. http://tcpreplay.appneta.com/, accessed 1 July 2015.

[4] AppNeta. Tcprewrite: Packet modifier, 2015. http://tcpreplay.synfin.net/wiki/tcprewrite, accessed 11 August 2015.

[5] Cisco Systems, Inc. Congestion Management Overview, 2014. http://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfconmg.html#wpxref23965, accessed 8 September 2015.

[6] L. Deri. nCap: Wire-speed packet capture and transmission. In *Proceedings of the Third IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON '05), Nice, 15 May*, pages 47–55. IEEE, 2005.

[7] Endace Ltd. Datasheet: Endace DAG 7.5G4 network monitoring card, 2015. http://www.emulex.com/

products/network-visibility-products-and-services/endacedag-data-capture-cards/specifications/, accessed 10 July 2015.

[8] GE Digital Energy. C60 breaker protection system, 2015. https://www.gedigitalenergy.com/multilin/catalog/c60.htm, accessed 5 August 2015.

[9] J. M. González and V. Paxson. pktd: A packet capture and injection dæmon. In *Proceedings of the Fourth Passive and Active Measurement Conference (PAM 2003)*, 2003.

[10] P. J. Hawrylak, J. Nivethan, and M. Papa. Automating electric substations using IEC 61850. In *Optimization and Security Challenges in Smart Power Grids*, pages 117–140. Springer, 2013.

[11] A. Heyde, L. Stewart, et al. Using the Endace DAG 3.7 GF card with FreeBSD 7.0. Technical Report 080507A, CAIA, May 2008.

[12] E. Hjelmvik. NetworkMiner packet analyzer, 2015. http://sourceforge.net/projects/networkminer/, accessed 11 August 2015.

[13] IEEE. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, July 2008. IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002).

[14] International Electrotechnical Commission. IEC 61850: Communication networks and systems in substations, 2004.

[15] International Electrotechnical Commission. IEC 61850: Communication networks and systems in substations—Part 5: Communication requirements for functions and device models, 2004.

[16] International Electrotechnical Commission. IEC 62351: Power systems management and associated information exchange—Data and communications security, 2007.

[17] J. F. Kurose. *Computer Networking: A Top-Down Approach Featuring the Internet, 3/E*. Pearson Education India, 2005.

[18] L. Martín Garcia. Programming with libpcap: Sniffing the network from our own application. *Hakin9*, 3(2):38–46, 2008.

[19] D. Miessler. A tcpdump primer with examples, 2015. https://danielmiessler.com/study/tcpdump/, accessed 1 July 2015.

[20] D. Mills, J. Martin, J. Burbank, and W. Kasch. Rfc 5905: Network time protocol version 4: Protocol and algorithms specification. *Internet Engineering Task Force*, 2010.

[21] Myricom. Sniffer10g cybersecurity software, 2015. http://www.myricom.com, accessed 2 July 2015.

[22] J. Nathan. Nemesis, 2004. http://nemesis.sourceforge.net/, accessed 2 July 2015.

[23] S. U. Rehman, W.-C. Song, and M. Kang. Network-wide traffic visibility in OF@TEIN SDN testbed using sFlow. In *Proceedings of the 16th Asia-Pacific Network Operations and Management Symposium (APNOMS), 17–19 September*, pages 1–6. IEEE, 2014.

[24] C. Sanders. *Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems*. No Starch Press, 2007. ISBN: 978-1-59327-149-7.

[25] Schweitzer Engineering Laboratories, Inc. SEL421 protection, automation, and control system, 2015. https://www.selinc.com/SEL-421/, accessed 5 August 2015.

[26] Tekron International. GNSS Timing Generator—TCG 01-G, 2015. http://www.tekron.com/tcg-01-g, accessed 5 August 2015.

[27] R. Van Der Knijff. Control systems/SCADA forensics, what's the difference? *Digital Investigation*, 11(3):160–174, 2014.

[28] Wireshark. Editcap: Edit and/or translate the format of capture files, 2015. https://www.wireshark.org/docs/man-pages/editcap.html, accessed 11 August 2015.

[29] W. Wu, P. DeMar, and M. Crawford. Why can some advanced Ethernet NICs cause packet reordering? *IEEE Communications Letters*, 15(2):253–255, 2011.

[30] B. Zhu, A. Joseph, and S. Sastry. A taxonomy of cyber attacks on SCADA systems. In *Proceedings of the IEEE International Conference on Internet of Things and Cyber, Physical and Social Computing (iThings/CPSCom 2011), 19–22 October, China*. IEEE Computer Society, 2011.